Lecture #01

Date: Aug-18,25 Day: Mond

Topic : Software Development Life cycle (SDLC)

1- Requirements (Analysis)
2- Planning (i.e., efforts, time, resources)
3- Design
4- Implementation (programming)
5- Testing (Q/A)
6- Deployment (Installation)
7- Maintainance
   → Extension
   → Changes
   → Upgrades → Bugs

Book : Object Orientated Software-Engineering by Lethbridge and Legnaire.

Lecture #02

Date: Aug 20, 2025 Day: Wednesday

Characteristics of Good Programs:

1- Readability
2- Writability (writing shorter code for a specific task)
3- Abstraction (A good abs. hides implementation details) * provides easy to use interface
4- Reuseability
5- Reliability
6- Maintainability (easy to maintain)
7- Modularity

* In distributed applications : Java is more writable than C++.

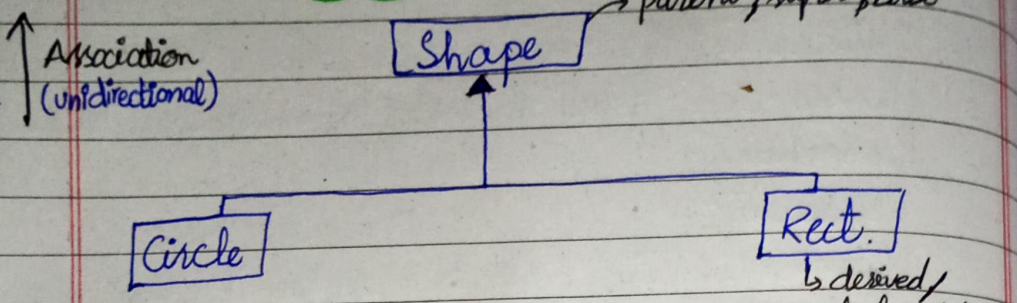
Selection Sort

```
void sort (int a[], int n) {

    for(int i = 0; i < n ; i++) {
        int m = min(a, i, n-1);

        swap(a[i], a[m]);

    }
```

Lecture # 03

Date: Aug 25, 25 Day: Monday

Inheritence



"Circle is a shape".

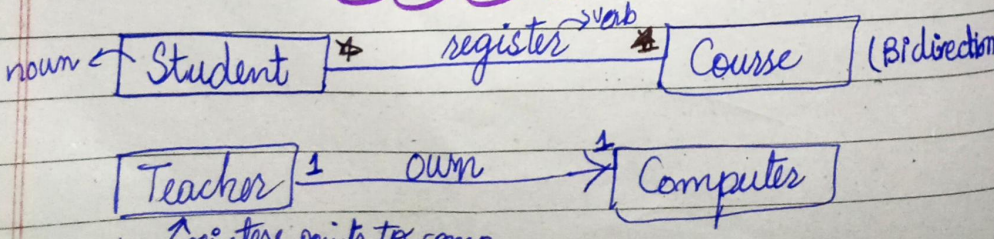
Polymorphism

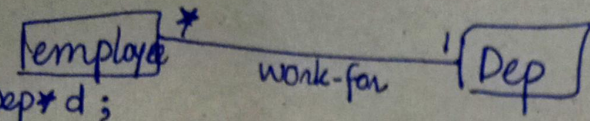
"A single line of code may invoke / call different functions (at diff. times).

```
void updateView (shape * a[], int n){
    for(int i = 0; i < n; i++)
        a[i] -> draw();

}
```

// if polymorphism doesn't exist: have to ~~diff~~ send different arrays for rectangle or circles or etc

Association





Date: Day:

→ Multiplicity : (1-1) (N-1) (N-M)

or

→ Cardinality :

: (1-1) bidirectional

```
class Teacher {
    string name;
    string address;
    ------
    Computer *c;
}
```

```
class Computer {
    string brand;
    string model;
    ------
    Teacher *t;
```

```
int callPay(Emp* e, char type){
    switch(type){
    case 's':
        return e->sal;
    case 'h':
        return (e->rate) * (e->hrs);
    case 'c':
        return (e->rate)/100 * (e->sales);
    }
}
```

```
class employee {
```

Date:

```
class Employee{
    string name;
    string address;
virtual int calculateSalary(){ }
};
```

```
class SalaryEmployee : public employee{
    int salary;
//constructors---
    int calculateSalary(){
        return salary;
    }
};
```

```
class hourlyBased : public Employee(){
    float rate;
    int hours;

int calculateSalary(){
    return this->rate * this->hours;
}
};
```

```
class commissionBased : public Employee()
    float rate, sales;

int calculateSalary(){
    return rate / 100 * (e->sales);
}
```

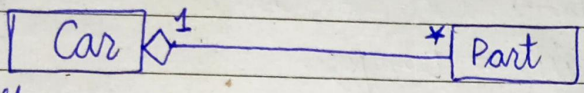
Lecture #04

Date: 27 Aug 25 Day: Wednesday

Topic: Types of Association

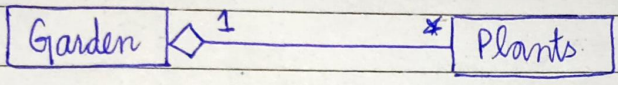
1. Simple Association:
→ pointer as a data member

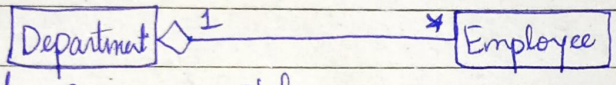
2. Aggregation:
→ Stronger kind of association.
→ Part-whole relationship, or containment or ownership.



"◇→ on the side of container"

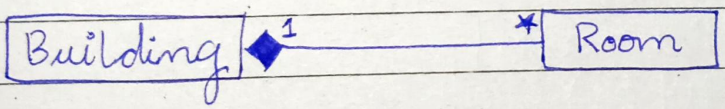
• Mostly one to many.

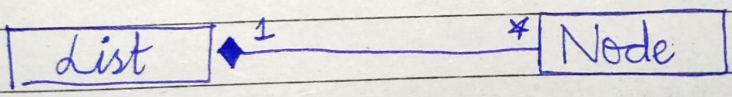




→ sharing possible.

3. Composition:
→ memory allocation is in the class.
→ Stronger kind of aggregation.
→ Life time binding
→ No sharing (Mutual Exclusion)





"if the data member of the class is itself a class"

```
class list{}{
    node *head, *tail;
~List(){
    for(node *ptr=head;---)
        delete ptr;
```

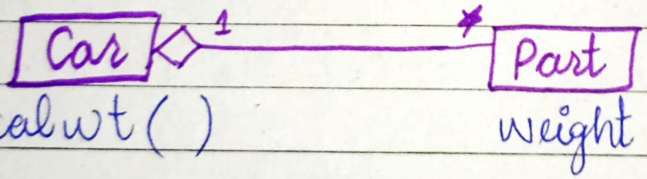
Date: method in the class (Linked list ka class ka function) Day:

```
void add(int x){
    node * temp = new node(x);
    tail -> next = temp;
    tail = temp;
}
```
// Composition

// not composition → can be shared into another class if we put this ptr in another class function.

```
void add (node * temp){
    tail -> next = temp;
    tail = tmp;
}
```
∴ there is sharing

// There is no diff b/w the codes for the association and aggregation. They are just concepts for design.




```
class Car {
    Part * a[N];
int callWt(){ int s=0;

for(i=0; i<N; i++){
    s = s + (a[i] -> getwt());
}
return s;
}
```